Linguagens de Programação

Fabio Mascarenhas - 2015.2

http://www.dcc.ufrj.br/~fabiom/lp

Substituição sem captura

- Quando substituímos um nome **x** por uma expressão **es** em outra expressão **ec** que introduz um nome **y**, pode ocorrer uma captura do nome **y**, caso **y** seja livre em ec

```
let _x = y + 2 in
  let y = 5 in
    _x + y
  end
end
```

captura!

- Nesse caso, o correto é renomear y por algum nome único, trocando todas as referências a y em ec pelo novo nome

```
let _x = y + 2 in
  let y = 5 in
    _x + y
  end
end
```

```
let z = 5 in
  (y + 2) + z
end
```

captura evitada

Funções de primeira classe

- Vamos adicionar funções anônimas a *fun*, e uma forma de chamá-las:

```
exp : ...
    | FUN '(' params ')' exp END
    | '(' exp ')' {'(' exps ')'}
```

```
case class Fun(params: List[String], corpo: Exp) extends Exp
case class Ap(fun: Exp, args: List[Exp]) extends Exp
```

- Uma função agora pode ser o valor de uma expressão, então também precisamos de:

```
case class FunV(params: List[String], corpo: Exp) extends Valor
```

Funções de primeira classe

- O resultado de avaliar uma `Fun` é um `FunV`
- Para avaliar um `Ap(fun, args)`, avaliamos `fun` esperando obter um `FunV(params, corpo)`, e aí a avaliação segue como a de `Ap1`
- Na avaliação small-step, uma Fun dá um passo para ela mesma, `Ap(Fun(param, corpo), args)` dá um passo similar ao de `Ap1`, e `Ap(fun, args)` dá um passo em `fun`
- Variáveis livres e substituição de `Fun` são parecidas com o `Let` (inclusive no cuidado com captura), e as de `Ap` são parecidas com as de `Ap1`

Funções de primeira classe

- Variáveis livres em uma função no momento da avaliação podem causar problemas
- Podemos ter captura indevida na substituição mesmo para variáveis call by value
- O exemplo à esquerda não tem problemas, mas o da direita precisa evitar a captura do `a` do `Let` pelo `a` da função `f`

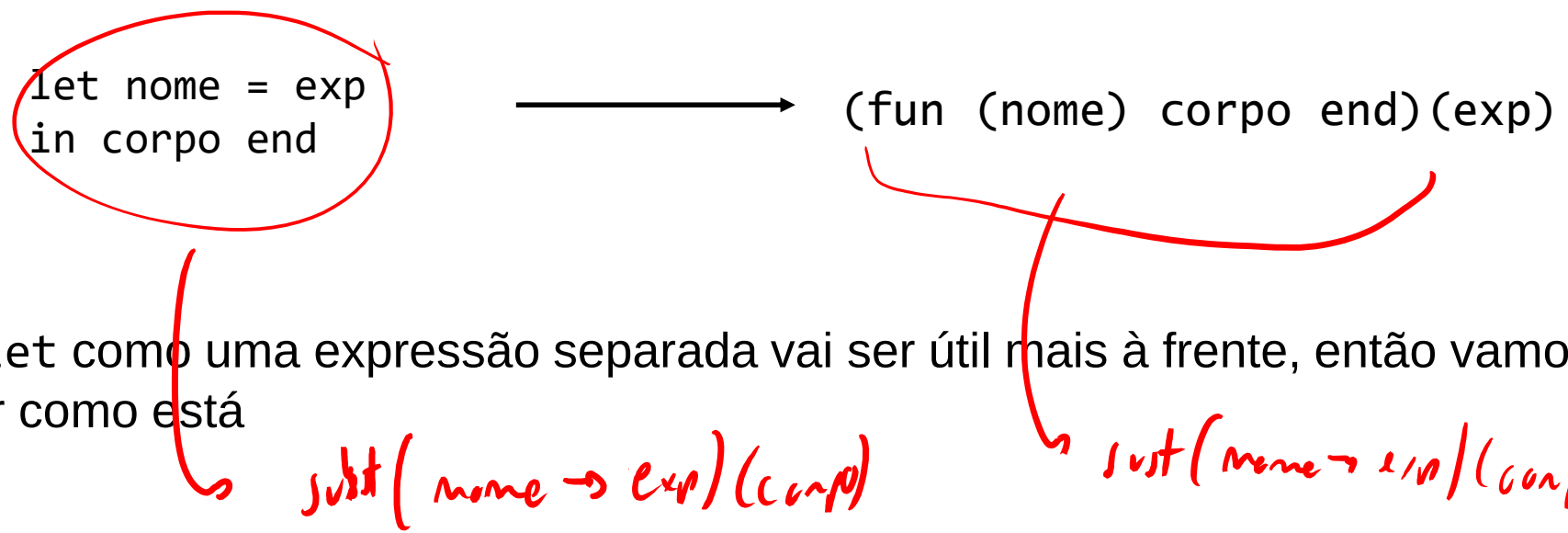
```
fun soma(a)
  fun (b)
    a + b
  end
end

(soma(2))(3)
```

```
let f = fun (b) a + b end in
  let a = 2 in
    (f)(a)
  end
end
```

Let é açúcar sintático

- As semelhanças entre `fun` e `let` não são coincidência

- Com funções de primeira classe, o let pode virar açúcar sintático para a definição e imediata aplicação de uma função de anônima:



- Mas `let` como uma expressão separada vai ser útil mais à frente, então vamos deixar como está